Master-Slave Control structure for massively parallel System on Chip

Hana Krichene, Mouna Baklouti, Jean-Luc Dekeyser, Philippe Marquet, Mohamed Abid

2013 16th Euromicro Conference on Digital System Design

Master-Slave Control structure for massively parallel System on Chip

Hana Krichene
LIFL - INRIA Lille-North Europe labs
University of Lille 1
Lille, France
National School of Engineers of Sfax
CES lab
University of Sfax
Sfax, Tunisia
Email:hana.krichene@inria.fr

Mouna Baklouti
and
Mohamed Abid
National School of Engineers of Sfax
CES lab
University of Sfax
Sfax, Tunisia
Email: mouna.baklouti@enis.rnu.tn
mohamed.abid@enis.rnu.tn

Philippe Marquet
and
Jean-Luc Dekeyser
LIFL - INRIA Lille-North Europe labs
University of Lille 1
Lille, France
Email: Philippe.Marquet@lifl.fr
jean-luc.dekeyser@lifl.fr

***Abstract*—The performance of massively parallel processing system depends mostly on the control configuration that is inherently part of the system. In particular, centralized control configuration is rigid and limits system scalability, and distributed control configuration is difficult to control in processing elements (PEs) interaction. Maintaining a flexible autonomous computation coupled with regular synchronous communication can assure a efficient parallel processing. The master-slave control structure is specified in such a way that previous features of the massively parallel System-on-Chip (mpSoC) are preserved and performance is improved. In this paper, we define the prototyping of a master-slave control structure for mpSoC in a FPGA-based platform. The structure implementation and related experiments using the vhdl language running on virtex6 ml605 of Xilinx board are described.**

I. Introduction

By the end of the eighties, massively parallel processing computers were much known in the community as high performance machines, especially in term of computing speed. Most of those machines can be classified into two general categories based upon the number of instruction streams active concurrently with the computational engine. Those parallel processing systems that execute a single thread of control are labeled Single Instruction, Multiple Data (SIMD) machines. Those that have the capability of executing many separate threads of control are called Multiple Instructions, Multiple Data stream (MIMD) machines. Despite their success and their suitability for a class of applications, the popularity of the SIMD machine is decreasing because of its rigidity mainly due to the centralized control which allows only synchronous broadcast and execution of each instruction. Otherwise, the distributed control in MIMD, makes the task of processors interaction difficult to manage. To broaden the applications scope, mixed-mode parallel processing systems add a new dimension in that they are capable of executing instruction in both SIMD and MIMD modes of parallelism and can switch between the two modes at instruction level granularity. But when using classic control system to switch from MIMD to SIMD mode, some processors may remain idle while they wait for the other processors to reach the switch point. This overhead and this difficult inter-PEs coordination control can degrade the performance of mixed-mode systems.

Nevertheless, nowadays, many modern application domains are concerned by the conjunction of regular parallel algorithms and high computing resources. They include signal and image processing applications such as software radio receiver, sonar beam forming, or image encoding/decoding. Furthermore, the implementation of the system on a single chip will be of prime interest for those applications that also require some degree of embeddedness. Present massively parallel System-on-Chip (mpSoC) have demonstrated their suitability for these modern applications but they are still limited by their rigidity due to classic control configuration, which is the inherently part of a mpSoC system. Therefore, the implementation of the control structure that coordinates the use of shared resources, must be carefully designed to avoid limiting system scalability. We define the master-slave control structure, which provides parallel local control when independent parallel execution is possible and global coordination when global synchronization is needed. This solution gives more execution flexibility and improves the scalability of systems, which can handle hundreds or even thousands of independent Processing Elements (PEs).

The objective of this paper is to reconsider the interest and the feasibility of a master-slave control structure into massively parallel architecture, especially in the context of single chip system integration. The goal of this structure is to provide in a single System-on-Chip two levels of control units: *the Master*, which controls system execution and allows synchronous communication and *the array of slaves*, which achieves parallel multifunctional control: parallel PE activities, parallel inter-PE communication, parallel synchronization, and parallel instructions execution. This structure allows autonomous processing with simple and regular communication, which improve the system execution time. We provide an RTL version that leads to a physical realization onto FPGA.

The next section presents some significant works related

`/13 $26.00 © 2013 IEEE
DOI 10.1109/DSD.2013.103

to control structure in massively parallel systems. Section 3 introduces the master-slave control paradigm. The design of SCU controller is described in section 4. The structure implementation and the performance evaluation are presented and discussed in section 5.

II. Related works

A small number of projects over the last thirty years used the idea of a hierarchical control structure for parallel computer. The EGPA project [1] was based on a pyramidal hardware hierarchy. The processors on the bottom layer were connected in a square mesh pattern and executed user code, while higher levels were used by the operating system. However, it was never extended beyond two levels, and no operating system algorithms were published for it.

Systems like Micros [2] and Chopp [3] used the software approach, creating a dynamic virtual hierarchy as needed. Thus, some of the processors were used for control, and those left over were available for users. These hierarchies were only used to map groups of processes to processors, thus partitioning the machine. Once a group of processes was mapped, it was executed without preemption until all processes terminated. Therefore, these systems were not interactive.

Several prototypes of mixed-mode (SIMD/MIMD) systems, e.g., PASM [4], TRAC [5] and OPSILA [6], were implemented before the mid 1990s using a static two-level hardware control structure. In these systems, the processors serving as control units and the computing PEs were most often exclusively defined at static time and there was an interconnection network between these two groups. Thus, the sole assignment of a processor either as a control unit or a PE was fixed throughout execution. This can cause performance degradation due to inefficient utilization of the available resources. Hence, they are less scalable and the partitioning is also not optimal.

Fifteen years after the decline of traditional massively parallel systems, significant evolutions of system design, silicon integration technology and growing application computing power requirement have change the context and it clearly seems important to consider and verify the feasibility and performances of massively parallel machines on a chip. We note the Hierarchical SIMD (H-SIMD) [7] System-on-Chip with hierarchical control composed of three layers: the host controller, the FPGA controller and nano-processor controllers. The switch between pairs of data memory banks overlaps operand communications with computations, thus hiding communication overheads to improve performance. The limit of this architecture is that the external host controller requires additional hardware resource other than the FPGA platform.

Recent years have seen related research known by mixed-mode HERA system on chip [8]. Every PE in HERA is equipped with its own control unit so that not only the whole system is dynamically partitionable, like previous mixed-mode systems, but also the role of each PE can be changed dynamically at runtime by using an HERA instruction, as needed. Although, the flexibility and the performance of this system, the switching mode and PEs communications seem to be tedious and time consuming.

Our purpose here is to present the new master-slave control approach which allows switched execution mode in the same parametric and modular hardware implementation. This structure is designed into mpSoC where each PE executes parallel instructions and then synchronously communicates with its neighbors via regular X-net network. The key features include: mask identification, node activity, autonomy execution, regular communication and barrier synchronization. All these features are controlled by the SCU component.

III. Master-Slave control structure

Generally speaking, the control of resources in a massively parallel system can be centralized or distributed, or some combination of the tow. Specifically, the following configurations have been proposed:

1) Centralized configuration in SIMD system: In this approach, the operating system runs on the master processor and broadcasts parallel instructions to the PEs. The master is a critical component in such systems; if it becomes overloaded, the whole system is affected. As a consequence, this design has limited scalability. In addition, at each clock cycle, each PE executes the same instruction under the direction of the master in lockstep manner. This synchronous execution induces the system rigidity.
2) Distributed replicated configuration in MIMD system: In this decentralized approach, the operating system runs on each processor with full copy of its data structures. This avoids the critical control of master processor but places a large demand on storage space. Also, the key point in parallel computing is that the processors can interact easily, and such interaction is difficult to control in a loosely coupled system.

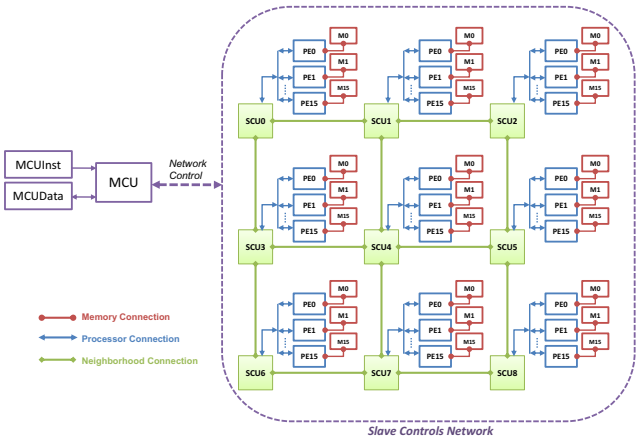

Fig. 1. Hardware prototype of Master-Slave control structure for mpSoC

A novel control structure is proposed for the massively parallel System-on-Chip, referred to as master-slave control. Its concept departs from the first configuration, the centralized

configuration. However, instead of a uni-processor master controlling a set of parallel PEs, the master cooperates with a grid of parallel slave controllers which supervises the activities of cluster of PEs. We define, as shown in fig 1, the hardware implementation of this configuration in massively parallel system:

- The Master Control Unit (MCU), which controls the order execution in the whole system. It is a simple processor, which fetches and decodes program instruction and broadcasts execution ordres to Slave Control Unit. It control the end execution to establish synchronous communication.
- The Slave Control Unit (SCU), which controls: local node and PEs activities, parallel instructions execution and synchronous communication. It is a crucial component in the master-slave control structure. The SCUs grid allows independent parallel execution.

The hardware architecture is composed of a single MCU and multiple Slave controllers (SCUs) combined with local processing element (PE) (or a cluster of 16 PEs), known collectively as Nodes. The MCU and SCU array are connected through single level hierarchical bus and the SCUs are connected together through X-net interconnection network. This network is clocked synchronously with the SCUs and respectively with the PEs. SCU controllers in the grid care for the instruction execution activities that involve a large degree of parallelism and the communication activities that need to coordinate all the PEs in the grid. Note that the SCU controllers do not limit performance; they do not become a sequential bottleneck. They participate only in the controlling and scheduling of very large groups of PEs execution at a time. Their functionality will be detailed later.

The idea of master-slave control should be distinguished from other hierarchical or clustered approaches proposed for parallel computing. Such proposals are usually motivated by memory latency considerations and the desire to build a scalable system. The use of two control levels is therefore visible to the user in its effect on the communication between various processors. With master-slave control structure, the PEs in massively parallel system can execute independently and then can communicate synchronously. Such a construction has the advantage of allowing the designer to optimize distinct processors for their intended tasks and to implement simple interconnection network without additionally buffers and complex routing algorithms. Separate communication phase from computation one, not only allows structural and regular processing, but also gives the flexibility on the choice of computation mode (SIMD or SPMD) in the same hardware implementation, which improve the system execution time.

IV. Design of Master-Slave Control Structure

Distinguish computation stage of that of communication needs the separation of these two stages in different blocks. This repartition should be provided by the designer at programming level. Then, the execution of computation blocks will be done in SIMD (by sending the parallel instruction) or SPMD (by sending the reference address of local memory program) computation mode according to the program description. In fact, operating system must decide not only which program executes when, but also where. Specifically, we can allocate groups of PEs to distinct parallel instruction blocks that run side by side, rather then always giving the whole system to one program and switching from one program to another. To perform this feature while avoiding the centralized control, we define second control level, other than the MCU, achieved by the SCU component.

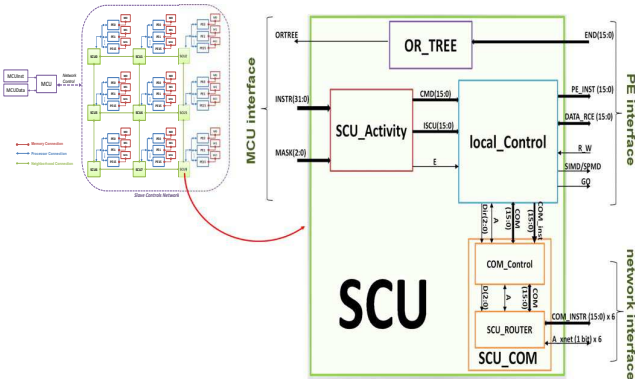

Fig. 2. Architecture of the Slave Control Unit (SCU)

A. SCU architecture

The SCU is the major component in the master-slave structure. It is composed of 4 connected modules, as shown in fig. 2, to control four main mechanisms:

- Local activity: achieved by the SCU_Activity module. It controls the local activity through the broadcast with mask mechanism [9]. Only activated SCUs receive instructions from the MCU while the others are in the idle state;
- PEs activity/parallel execution: achieved by the local_Control module. It controls the activity of PEs involved in the processing and the execution of parallel instructions;
- Regular communication: achieved by the SCU_COM module. It controls the inter-SCUs data transfer using communication instructions, which specify the communication parameters (direction and distance);
- System synchronization: achieved by the OR_Tree module. It is a barrier synchronization that allows the SCU controller to always know if all activated PEs achieved the processing.

These mechanisms will be presented in details through the next sections.

1) SCU-activity: The overhead due to the "mask" construction involves the time spent by the Control Unit in determining, which PEs should be enabled/disabled in the grid. To reduce the effects of this overhead, master-slave control achieves massively parallel masking operations in two steps using the broadcast with mask [9] mechanism through the SCU-activity

module. First, it identifies active nodes according to a specific mask. In fact, using the mask instructions shown in table I, this module sets to 1 the bit flag (BF) register inside each node involved in the processing. Consequently, only nodes with BF register set to 1 will receive parallel instructions. Second, when the mask is mapped onto the SCUs-grid, the SCU-activity module decodes broadcast instructions, which are presented in table I. According to these instructions, the MCU broadcasts parallel execution orders to selected nodes while the mask is not changed. Each active SCU receives 32 bits parallel instruction and then sends the first sixteen most significant bits (MSB) to CMD output (it represents the control SCU instructions) and the last sixteen least significant bits (LSB) to ISCU output (it represents the SIMD instructions or SPMD reference address for PEs).

TABLE I
MASK AND BROADCAST INSTRUCTIONS SET

Mask Instructions		
selbf	(000)	activate SCUs in the selected set
selbfand	(001)	activate SCUs in the intersection of sets
selbfor	(010)	activate SCUs in the union of sets
selbfxor	(011)	activate SCUs in the union of sets except the intersection part
Broadcast Instructions		
brdbf	(100)	broadcast parallel instructions to active SCUs
brdbfb	(101)	broadcast parallel instructions to inactive SCUs
brdall	(111)	broadcast parallel instructions to all the SCUs

We notice that the use of SCU-activity module in the master-slave control structure allows the sub-netting of the SCU grid which optimizes the data flow transfer and increases the parallel broadcast domains. This control of the network traffics improves the power allocation to the processing nodes. After sub-netting the network, the parallel execution orders will be broadcasted to local-control module, which controls the computation and communication phases. This module will be detailed in the next section.

2) Local-control: The overall structure of the local-control is shown in fig. 3. The main sub-module in local-control is the Instruction Decoder, which is the master of inter-nodes communication and cluster PEs activities. First, it selects the computation mode (SIMD or SPMD) and activates the PEs according to the local mask. It transfers data between SCU/PEs or inter-SCUs in the network, if it is requested, and controls PEs instructions execution through the tree of end signals. The switch between the two modes of parallelism and the control of computation/communication in cluster of PEs are done in a single clock cycle at instruction level granularity as defined in Table II.

The local-control defines specific registers:

- R_COM, which contains data to be sent into the inter-connection network;
- R_ISCU, which contains reference address or parallel instruction to be sent to the active PEs in the cluster;
- R_SCUi and R_PEi, which serve as temporary memory for SCU data and PE data, respectively.

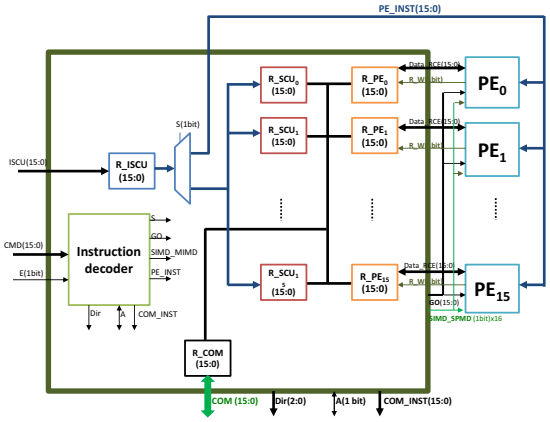

Fig. 3. Architecture of the local-Control

The data busses in this module are 16 bits wide and each instruction is a 16-bits word.

The simplicity of the PEs activities and the decentralization of parallel execution control, through the local-control module, makes the task of the selection of execution modes (SIMD/SPMD) easy and flexible in the same mpSoC architecture. These features improve system scalability to fit modern application requirements.

3) SCU-COM: The SCUs components in mpSoC system are connected in two-dimensional mesh to form the X-net interconnection network via specific communication module called: SCU-COM. In fact, SCU-COM module at each SCU component allows communications with any of 8 neighbors using only 6 wires per component. Each SCU-COM has two hierarchical routers presented in the fig. 4:

(a) R-SCUXnet with 5 connections: 4 at its diagonal corners and one connected to the COM-Control module, which manages directions and distances of communications;
(b) R-Xnet with 4 connections at its diagonal corners, forming an X pattern.

Each Xnet router can take 4 different directions: North West, North East, South West, South East. But, the data should be able to move in 8 directions. This is done with the couple (”R-Xnet” , ”R-SCUXnet”), which receives the specific direction from COM-Control and then calculates the one to take.

In some cases, the connections at the SCU array edges are wrapped around to form a torus, which facilitates several important matrix algorithms. All SCUs have the same direction controls so that, for example, every NE-COM sends an operand to the North and simultaneously receives an operand from the South. The X-Net uses a bit-state implementation, clocked synchronously with the SCUs to identify nodes, which participate in communication; all transmissions are parity checked. Inactive SCU-COMs can serve as pipelines stages to expedite long distance communication jumps through several SCU-COMs. This transfer of data occurs without conflicts.

TABLE II
SCU INSTRUCTIONS

	Name					Function
	CMD(16bits)				ISCU(16bits)	
	Opcode(6bits)		(5bits)	(5bits)		
SPMD Instruction	TAC_IND	(000000)	R_SCUi		address	Activate PEs according to the mask in R_SCUi register
	TAC_ALL	(000001)				Activate all PEs
	TIC_IND	(000010)	R_SCUi		address	Deactivate PEs according to the mask in R_SCUi register
	TIC_ALL	(000011)				Deactivate all PEs
SIMD Instruction	SIMD_IND	(000100)	R_SCUi		P_INST	Activate PEs according to the mask in R_SCUi register
	SIMD_ALL	(000101)			P_INST	Activate all PEs
Transfer Register	MOV	(000110)	R_SCUi/R_PEi	R_SCUj/R_PEj		Move R_SCUj/R_PEj register in R_SCUi/R_PEi register
	MOV_I	(000111)	R_SCUi		value	Move value in R_SCUi register
Communication Instruction	MOV_R_COM	(001000)	R_SCUi			Move R_COM register in R_SCUi register
	MOV_W_COM	(001001)	R_SCUi			Move R_SCUi register in R_COM register
	COM_S	(001010)	DIR	DIST		SEND data from R_COM register to network
	COM_R	(001011)	DIR	DIST		RECEIVE data from network to R_COM register

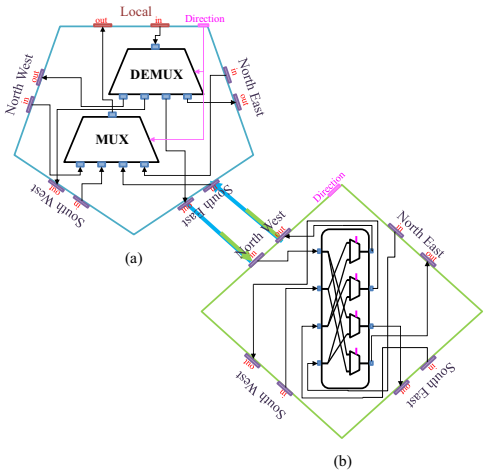

Fig. 4. Architecture of the SCU-Router

Sending and receiving data through networks are managed by different communication instructions. A given Xnet communication allows all the SCUs to communicate together in a given direction at a given distance through the SCU-COMs modules. Direction and distance are here the same for all the SCUs. Such communication is realized in several communication phases driven by the SCU through the local-Control module that decodes the communication instructions and then sends the appropriate micro-instructions (SEND_XNET and RECEIVE_XNET) to the SCU-COMs modules to establish the communications links.

Through this regular and modular interconnection network, synchronous communications can be achieved via simple routing design without additionally buffers and complex routing algorithms.

4) OR-Tree: The Barrier synchronization is a high latency operation and a number of machines have proposed or implemented fast barrier mechanisms in hardware. Several systems have implemented either dedicated barrier networks [10], [11] or provided hardware support within existing data networks [12], [13]. The OR-Tree is a mechanism for global OR; serves as a test for checking the state of the network of nodes. It is performed by a tree of "OR" gates which compares the end execution signals of all the nodes in pairs. It is a barrier synchronization that allows the controllers to know if all activated PEs finished the computation. The master-slave control structure supports two hardware barrier mechanisms. The first one integrated in the SCU component to test the end execution in cluster of PEs and the second one in the SCU-grid to test the end execution in the SCU network.

```
Architecture structural of OrTree is
signal lout, rout: std_logic;
begin
baseCase     : if width = 1 generate
                    o <= ins(0);
               end generate;
recursiveCase: if width > 1 generate
                  left : entity OrTree
                         generate map (width/2)
                         port map (lout, ins(width/2 downto 0));
                  right: entity OrTree
                         generate map ((width+1)/2)
                         port map (rout, ins(width-1 downto width/2));
                  joint: o <= lout or rout;
               end generate;
end structural;
```

Fig. 5. Structural architecture of an OR-Tree in VHDL

Using a combination of conditional and recursive instantiation, a structural architecture for OR-Tree is defined in fig. 5. In fact, The OR-Tree module takes a generic parameter width, a compile-time value specifying how many inputs the OR-Tree operates on. It also takes an array of inputs, width elements long, and produces a single output representing the logical disjunction of all the inputs. Using a combination of conditional and recursive instantiation, a structural architecture for OR-Tree is defined.

V. EXPERIMENTAL RESULTS

A. Hardware cost

Using the IP blocks of master-slave control structure in mpSoC, it was easy to prototype different configurations on the FPGA Virtex6 ML605 device. The table IV shows some

synthesis results varying the mpSoC parameters as well as its integrated components. The processor used in these designs is the forth [14].

TABLE III
SYNTHESIS RESULTS ON VIRTEX6 ML605

Logic Utilization		
Module	LUTs	Registers
SCU-Activity	4	2
local-Control	652	304
SCU-COM	202	50
OR-TREE	3	0

Almost the entire cost of providing master-slave control structure is silicon area used by the SCU component. The SCU contains four control modules, which are likely to be small as shown in table III. In fact, table IV shows that for 100 nodes with 2-bytes instructions buses on the current Virtex-6 ML605, having 100 PEs with 4KB memory each one, the SCUs occupy about 38% total consumed on-chip logic area; For 16 PEs, with 4KB per PE, it is around 16%. These numbers are large, but as feature size decreases, the incremental cost of adding SCU functionality to mpSoC control system quickly becomes small.

Memory still a critical component in the context of mpSoC scalability. Indeed, the challenge for massively parallel on-chip implementation is the reduction and optimization of memory allocation. This problem does not arise in this case because the consumption of memory blocks does not exceed 13%. In addition, comparing different configurations, we note that although it is expensive on surface occupation, the mpSoC with master-slave control is not enormously power consumer.

We clearly notice that the mpSoC with master-slave control structure consumes more FPGA area than the simple mpSoC system. Thus, depending on his needs the designer can integrate the needed components (interconnection network, number of SCU controllers and number of PEs per cluster) in the selected mpSoC configuration. The use of reconfigurable IP blocks significantly facilitates the monitoring of processing nodes with rapid modification of system configuration. Consequently, the master-slave control permits this system to be flexible, scalable and easily tuned according to the application requirements.

B. Validation

What performance improvement may be expected by adding the master-slave control to mpSoC ? This section gives better insight on the performance of the proposed flexible mpSoC system. The fig. 6 presents the execution time results running FIR filter application on mpSoC using master-slave control and compared to other systems using centralized (SIMD system) or distributed (MIMD system) controls. The result is based on an impulse response with a length of four, which in the chosen implementation requires four multiplications and three additions per output signal. The rest of the instructions are overhead in forms of communication and memory instructions.

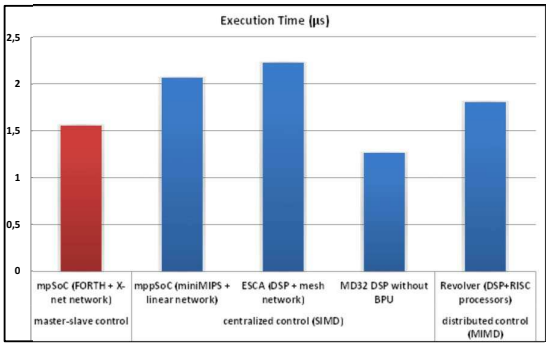

Fig. 6. Execution time of FIR filter processing

From these experiments we demonstrate the effectiveness of mpSoC with master-slave control to compute FIR application. We notice that this mpSoC architecture allows more rapid processing than mppSoC [15] and ESCA [16] SIMD architectures. This can be explained by the use of a simple planar and modular design in our mpSoC architecture. In fact, the use of a master-slave control decentralizes system monitoring, which allows autonomous and rapid processing.

In addition, the integration of reconfigurable and regular X-net network supplies reliable synchronous communication through simple interconnection without additionally buffers and complex routing algorithms. With this network, independent PEs can interact easily opposed to MIMD architecture where PEs coordinations are difficult to control. This can explain why the overall execution time in our mpSoC system is less than in Revolver architecture [17].

We notice comparable results between our mpSoC execution and the DSP [18] execution. This is explained by the fact that the DSP is realized on an ASIC and presents a higher frequency than the FPGA based mpSoC with master-save control. Nevertheless, the DSP is slightly faster than our proposed SoC.

This analysis demonstrates the efficiency of master-slave control to compute data intensive processing application. In fact, it allows the PEs of our mpSoC to adopt a faster instructions execution than the PEs of existing massively parallel machine and provides parallel inter-nodes coordination when needed without compromising system speed-up. Giving these results, it is possible and convenient to increase the performance of massively parallel system, both from the technological and from the architectural point of view.

C. Evaluation

Performances. The master-slave control is interesting for general purpose parallel algorithms which require the simultaneous execution of a number of parallel tasks. It coordinates the partitioning and the control of resources among the running programs due to the changing needs of switched computation models. This structure permits the development of large, modular parallel systems by allowing the different modules to be programmed and executed as if they were independent

TABLE IV
SYNTHESIS RESULTS ON VIRTEX6 ML605

System	Number SCUs	Logic Utilization LUTs	Registers	%	Total Memory 18Kb Block RAM (4096 bytes RAM-PE)	%	Power Consumption (mWatts)
mpSoC with centralized control	16	19155	2209	14	8	1	2989
	64	81245	8823	56	32	7	3000
	100	135515	14309	93	55	13	3215
mpSoC with master-slave control	16	19282	2645	15	8	1	2987
	64	81544	11644	57	32	7	3179
	100	142887	23222	97	55	13	3329

jobs. The mpSoC using this technique scales very well, and its use dramatically decreases the total execution time because the synchronization is needed only in the communication stage which allows a simple system control with a single clock domain (one distributed clock). In this last case, this becomes clearly an advantage for data parallel applications on large scale where power efficiency is required. Nevertheless, be aware that there is a small area overhead when designing the master-slave control that needs extra-hardware components (the SCUs) in mpSoC system to switch and control computation modes.

Ease of use. The designer has to configure the parallel processing system and then broadcast parallel instructions or reference memory addresses to activated nodes according to the chosen computation modes (SIMD or SPMD). To better monitoring the computation modes in the same hardware implementation and coordinate between PEs without compromising scalability, the master-slave control structure is integrated onto an existing mpSoC architecture. So that, the designer does not need a new design to achieve previous features, but only adds control instructions as shown in table II. This ease of use is an argument in favor of a systematic usage of this control structure.

*Implementation.*The master-slave control structure is composed of two levels of reconfigurable IP blocks. In addition to the MCU, which only specifies the mask configuration and the computation mode, the grid of SCUs controls the PEs activities, the parallel instructions execution and synchronous data communication. Independent control and management for processing nodes not only gives more flexibility to the system, but also increase system scalability through the use of relatively simple and reconfigurable IPs. Such a structure is interesting to support parallel distributed computations.

VI. CONCLUSION

In this paper, we have defined master-slave control structure as a new concept in the field of massively parallel processing System-on-Chip. This structure avoids the limits of centralized control type systems, in which the master can become a bottleneck, and totally distributed systems, which lack global PEs coordination. Combining the goals of being scalable, and fulfilling the paradigm of simultaneous parallel executions, the master-slave control structure provides, in a potentially simple and straightforward manner, the best attributes of both the SIMD and SPMD modes of computation in a same on-Chip System. When it is actually implemented, we can carry out a full cost/performance analysis. The ideas presented here are a step towards building a new theory of massively parallel execution model based on Synchronous Communication Asynchronous Computation: SCAC model.

REFERENCES

[1] W. Handler, A. Bode, G. Fritsch, W. Henning, and J. Volkert, "A tightly coupled and hierarchical multiprocessor architecture," in *Computer Physics Communications*, Jul. 1985, pp. 87 – 93.

[2] A. van Tiborg and L. Wittie, "Wave scheduling - decentralized scheduling of task forces in multicomputers," in *IEEE Trans. Computers*, 1984, pp. 835 – 844.

[3] H. Sullivan, T. Bashkow, and D. Klappholz, "A large-scale, homogeneous, fully distributed parallel machine, ii," in *Proc. 4th Ann. Int'I Symp. Computer Architecture*, Mar. 1977, pp. 118 – 124.

[4] H. J. Siegel, T. Schwederski, W. G. Nation, J. B. Armstrong, L. Wang, J. T. Kuehn, R. Gupta, M. D. Allemang, D. G. Meyer, and D. W. Watson, "The design and prototyping of the pasm reconfigurable parallel processing system," in *Parallel Computing: Paradigms and Applications, International Thomson Computer Press*, London, UK, Jun. 1996, pp. 78 – 114.

[5] G. J. Lipovski and M. Malek, "Parallel computing: Theory and comparisons," in *John Wiley & Sons*, New York, USA, Jun. 1987.

[6] P. Duclos, F. Boeri, M. Auguin, and G. Giraudon, "Image processing on a simd/spmd architecture: Opsila," in *Ninth International Conference on Pattern Recognition*, Nov. 1988, pp. 430 – 433.

[7] Xu and Xizhen, "A hierarchically-controlled simd machine for 2d dct on fpgas," in *SOC Conference*, 2005, pp. 276 – 279.

[8] X. Wang and S. G. Ziavras, "Exploiting mixed-mode parallelism for matrix operations on the hera architecture through reconfiguration," in *IEEE Proceedings, Computers and Digital Techniques*, Jul. 2006, pp. 249 – 260.

[9] H. Krichene, M. Baklouti, P. Marquet, J. L. Dekeyser, and M. Abid, "Broadcast with mask on a massively parallel processing on a chip," in *High Performance Computing and Simulation (HPCS2012)*, Madrid, Spain, Jul. 2012, pp. 275 – 280.

[10] R. H. Arpaci, D. E. Culler, A. Krishnamurthy, S. G. Steinberg, and K. Yelick, "Empirical evaluation of the CRAY-T3D: a compiler perspective," in *The 22nd annual international symposium on Computer architecture (ISCA95)*, New York, NY, USA, May 1995, pp. 320 – 331.

[11] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, W. D. Hillis, B. C. Kuszmaul, M. A. S. Pierre, D. S. Wells, M. C. Wong-Chan, S. W. Yang, and R. Zak, "The network architecture of the connection machine CM-5," vol. 32(2), New York, NY, USA, May 1996, pp. 145 – 158.

[12] D. K. Panda, "Fast barrier synchronization in wormhole k-ary n-cube networks with multidestination worms," in *The 1st IEEE Symposium on High-Performance Computer Architecture (HPCA95)*, Washington, DC, USA, Jan. 1995, p. 200.

[13] S. L. Scott, "Synchronization and communication in the T3E multiprocessor," in *The 7th international conference on Architectural support for programming languages and operating systems:ASPLOS-VII*, New York, NY, USA, 1996, pp. 26 – 36.

[14] R. Haskell and D.M.Hanna, "A VHDL forth core for FPGAs," *Microprocessors and Microsystems*, vol. 28, pp. 115 – 125, Apr. 2004.

[15] M. Baklouti, "A rapid design method of a massively parallel system on chip: from modeling to fpga implementation," in *http://tel.archives-ouvertes.fr/tel-00527894/en/*, Dec. 2010.
[16] P. Chen, K. Dai, D. Wu, J. Rao, and X. Zou, "Parallel algorithms for FIR computation mapped to ESCA architecture," in *International Conference on Information Engineering (ICIE)*, Beidaihe, Hebei, Aug. 2010, pp. 123 – 126.
[17] J. Oberg and P. Ellervee, "Revolver: A high-performance mimd architecture for collision free computing," vol. 1, Vasteras, Aug. 1998, pp. 301 – 308.
[18] C. Xiaoyi, Y. Qingdong, and L. Peng, "Data bypassing architecture and circuit design for 32-bit digital signal processor," in *Journal of Electronics*, Nov. 2005.